example a ChangeManager 955 may need to login to the database to trap the events. The UserObject object encapsulates a string-based userID and the notion of Credentials. Each Platform implementation provides its own LocalUser object. Implementations provide a subclass of the credentials Object customized for the security requirements of their system; in the simplest case the credentials are a String password.

```
public class UserObject implements Serializable
{
  String mUsername;
  Object mCredentials;

  public UserObject(String username, Object credentials)
  {
    mUsername = username;
    mCredentials = credentials;
  }

  public String getUsername()
  {
     return mUsername;
  }

  public Object getCredentials()
  {
     return mCredentials;
  }
}
```

The Local object contains information about the object that the connector uses uniquely identify an object in the native system. It holds the following information about the object (1) ID: An opaque object identifier, and (2) aClass: the type or class of the object.

The LocalObjectID class is defined as:

```
public class LocalObjectID
{
  Object mID;
  Object mClass;

  public LocalObjectID(Object ID, Object aClass)
  {
    mID = ID;
    mClass = aClass;
  }
```

```
    public Object getID()
    {
        return mID;
    }

    public Object getObjectClass()
    {
        return mClass;
    }
}
```

Referring to Figure 10, an example of the operation of the above Interconnect services in which a purchase order is delivered from a Source site 1000 to a target SAP system 1005 utilizing the Interconnect Server 1010 is set forth. An Importer component 1015 resides on the target SAP system and the Requestor 1020, Monitor 1025, Event Manager 1030, Accessor 1035, and Transformer 1040 components reside on the Interconnect Server 1010. At step 1, At the Source site 1000, a Purchase order 1045 is generated and a "SabaInvoice" object is created. At step 2, the Purchase Order 1045 is saved. Because it needs to be synchronized with a remote system, this triggers a pre-registered ChangeManager event at the EventManager 1030. At step 3, the ChangeManager passes the unique id of the SabaInvoice to the Monitor 1025. At step 4, the Monitor 1025 instructs the Accessor 1035 to retrieve the SabaOrder in Interchange Format. At step 5, the Accessor 1035 retrieves the SabaInvoice in serialized, canonical XML format. This is an internal XML format that varies for each business object. Its essential feature is that it contains all relevant information about the PO in attribute/value format. Step 5 uses a standard method available for all SabaObjects.

The following example Local Format document is a sample SabaInvoice serialized into XML:

```
<?xml version="1.0" standalone="yes"?>
<SabaObjectSerialization xmlns:dt="urn:w3-org:xmldatatypes">
 <SabaObject type="com.saba.busobj.SabaInvoice" id="invce000000000001000"
status="new">
```

```
                <amt_paid dt:type="number">0.0</amt_paid>
                <other_charges dt:type="number">0.0</other_charges>
                <acct_id
            idref="http://spanuganti/interconnect/Saba/com.saba.interconnect.ObjectID@94902deb/206"/>
                <updated_by dt:type="string">uone</updated_by>
                <balance dt:type="number">425.0</balance>
                <updated_on dt:type="dateTime">2000-11-10 19:17:40.000</updated_on>
                <created_by dt:type="string">uone</created_by>
                <created_id
            idref="http://spanuganti/interconnect/Saba/com.saba.interconnect.ObjectID@170064/6"/>
                <inv_date dt:type="dateTime">2000-11-10 19:17:40.000</inv_date>
                <created_on dt:type="dateTime">2000-11-10 19:17:40.000</created_on>
                <split dt:type="string">domin000000000000001</split>
                <status dt:type="number">100</status>
                <time_stamp dt:type="string">20001110191739926 2</time_stamp>
                <flags dt:type="string">0000000000</flags>
                <invoice_no dt:type="string">001000</invoice_no>
                <currency_id
            idref="http://spanuganti/interconnect/Saba/com.saba.interconnect.ObjectID@14966/34"/>
                <total_charges dt:type="number">425.0</total_charges>
              </SabaObject>

              <SabaObject type="com.saba.busobj.SabaInvoiceItem" id="invit000000000001000"
            status="new">
                <order_item_id idref="ordit0000000000001060"/>
                <invoice_id
            idref="http://bnemazie/interconnect/Saba/com.saba.interconnect.ObjectID@c82f961c/101"/>
                <time_stamp dt:type="string">20001110191740614 5</time_stamp>
              </SabaObject>

              <SabaObject type="com.saba.busobj.SabaOrder" id="extor000000000001040"
            status="new">
                <city dt:type="string">Sunnyvale</city>
                <addr1 dt:type="string">Addr 11</addr1>
                <country dt:type="string">US</country>
                <shipped_amt dt:type="number">0.0</shipped_amt>
                <state dt:type="string">CA</state>
```

```
<discount dt:type="number">0.0</discount>
<updated_by dt:type="string">UONE</updated_by>
<order_no dt:type="string">001040</order_no>
<updated_on dt:type="dateTime">2000-11-10 19:13:19.000</updated_on>
<created_by dt:type="string">uone</created_by>
<created_id
idref="http://spanuganti/interconnect/Saba/com.saba.interconnect.ObjectID@170064/6"/>
<shipped_attn dt:type="string">test1 test1</shipped_attn>
<contact_id
idref="http://bnemazie/interconnect/Saba/com.saba.interconnect.ObjectID@c9162811/1"/>
<created_on dt:type="dateTime">2000-11-10 19:13:19.000</created_on>
<sold_by_id
idref="http://spanuganti/interconnect/Saba/com.saba.interconnect.ObjectID@170064/6"/>
<split dt:type="string">domin000000000000001</split>
<status dt:type="number">400</status>
<time_stamp dt:type="string">20001110917406145</time_stamp>
<company_id
idref="http://spanuganti/interconnect/Saba/com.saba.interconnect.ObjectID@94902deb/206"/>
<territory_id idref="terri0000000000000001"/>
<conf_type dt:type="number">0</conf_type>
<zip dt:type="string">94086</zip>
<account_id
idref="http://spanuganti/interconnect/Saba/com.saba.interconnect.ObjectID@94902deb/206"/>
<currency_id
idref="http://spanuganti/interconnect/Saba/com.saba.interconnect.ObjectID@14966/34"/>
<status_flag dt:type="string">2000200000</status_flag>
<total_charges dt:type="number">425.0</total_charges>
<children>
<SabaObject type="com.saba.busobj.SabaOrderItem" id="ordit000000000001061"
status="new">
<order_id idref="extor000000000001040"/>
<unit_cost dt:type="number">425.0</unit_cost>
<description dt:type="string">Inventory1</description>
<actual_qty dt:type="number">1</actual_qty>
<part_id idref="prdct000000000001022"/>
<pkg_item_id idref="ordit000000000001061"/>
<created_on dt:type="dateTime">2000-11-10 19:13:28.000</created_on>
```

```
<req_qty dt:type="number">1</req_qty>
<delivered_on dt:type="dateTime">2000-11-10 19:17:13.000</delivered_on>
<status dt:type="number">300</status>
<time_stamp dt:type="string">20001110191740 6145</time_stamp>
<Custom0 dt:type="string">Billed</Custom0>
<flags dt:type="string">0000000000</flags>
<total_cost dt:type="number">425.0</total_cost>
<item_typ dt:type="number">1</item_typ>
<billing_state dt:type="number">101</billing_state>
</SabaObject>

<SabaObject type="com.saba.busobj.SabaOrderItem" id="ordit000000000001060"
status="new">
<order_id idref="extor000000000001040"/>
<unit_cost dt:type="number">0.0</unit_cost>
<description dt:type="string">Default  Default</description>
<actual_qty dt:type="number">1</actual_qty>
<part_id idref="shpmd000000000000001"/>
<pkg_item_id idref="ordit000000000001060"/>
<created_on dt:type="dateTime">2000-11-10 19:13:27.000</created_on>
<req_qty dt:type="number">1</req_qty>
<delivered_on dt:type="dateTime">2000-11-10 19:17:13.000</delivered_on>
<status dt:type="number">300</status>
<time_stamp dt:type="string">20001110191740 6145</time_stamp>
<Custom0 dt:type="string">Billed</Custom0>
<flags dt:type="string">0000000000</flags>
<total_cost dt:type="number">0.0</total_cost>
<item_typ dt:type="number">6</item_typ>
<billing_state dt:type="number">101</billing_state>
</SabaObject>

</children>
</SabaObject>

<SabaObject type="com.saba.busobj.SabaInvoiceItem" id="invit000000000001001"
status="new">
<order_item_id idref="ordit000000000001061"/>
```

```
            <invoice_id
idref="http://bnemazie/interconnect/Saba/com.saba.interconnect.ObjectID@c82f961c/101"/>
            <time_stamp dt:type="string">20001110191740614 5</time_stamp>
          </SabaObject>

</SabaObjectSerialization>
```

At step 6, the Accessor 1035 then transforms the XML document into an Interchange document format. The Accessor 1035 accomplishes this by passing the source document and an XSL stylesheet to the Transformer 1040.

The following is a sample purchase order XSL stylesheet:

```
<!--COPYRIGHT NOTICE Copyright (c) 1997-2000 Saba Software Inc., 2400 Bridge
Parkway, Redwood Shores, California 94065-1166 USA. All rights reserved.-->
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output omit-xml-declaration="no" indent="yes" method="xml"/>
  <xsl:template match="SabaObjectSerialization">
    <SYNC_INVOICE_001>
      <CNTROLAREA>
        <BSR>
          <VERB>SYNC</VERB>
          <NOUN>INVOICE</NOUN>
          <REVISION>001</REVISION>
        </BSR>
        <SENDER>
          <LOGICALID/>
          <COMPONENT/>
          <TASK/>
          <REFERENCEID/>
          <CONFIRMATION/>
          <LANGUAGE/>
          <CODEPAGE/>
          <AUTHID>
            <xsl:value-of select="created_by"/>
          </AUTHID>
        </SENDER>
        <DATETIME qualifier="CREATION">
          <YEAR>
            <xsl:value-of select="substring(//created_on,7,4)"/>
          </YEAR>
          <MONTH>
            <xsl:value-of select="substring(//created_on,1,2)"/>
          </MONTH>
          <DAY>
            <xsl:value-of select="substring(//created_on,4,2)"/>
```

```
        </DAY>
        <HOUR/>
        <MINUTE/>
        <SECOND/>
        <SUBSECOND/>
        <TIMEZONE/>
      </DATETIME>
    </CNTROLAREA>
    <DATAAREA>
      <xsl:for-each select="//SabaObject[@type='com.saba.busobj.SabaInvoice']">
        <INVOICE>
          <INVDATE>
            <xsl:value-of select="//inv_date"/>
          </INVDATE>
          <CURRENCYID>
            <xsl:value-of select="//currency_id/@idref"/>
          </CURRENCYID>
          <INVNO>
            <xsl:value-of select="//invoice_no"/>
          </INVNO>
          <INVOICEID>
            <xsl:value-of select="@id"/>
          </INVOICEID>
          <TOTALCHARGES>
            <xsl:value-of select="//total_charges"/>
          </TOTALCHARGES>
          <ACCTID>
            <xsl:value-of select="acct_id/@idref"/>
          </ACCTID>
          <CREATEDID>
            <xsl:value-of select="created_id/@idref"/>
          </CREATEDID>
          <UPDATEDON>
            <xsl:value-of select="updated_on"/>
          </UPDATEDON>
          <ORDERID>
            <xsl:value-of select="order_id/@idref"/>
          </ORDERID>
          <BALANCE>
            <xsl:value-of select="balance"/>
          </BALANCE>
          <AMTPAID>
            <xsl:value-of select="amt_paid"/>
          </AMTPAID>
          <OTHERCHARGES>
            <xsl:value-of select="other_charges"/>
          </OTHERCHARGES>
          <STATUS>
            <xsl:value-of select="status"/>
          </STATUS>
          <FLAGS>
            <xsl:value-of select="flags"/>
          </FLAGS>
          <SPLIT>
            <xsl:value-of select="split"/>
```

```
                </SPLIT>
                <POID>
                  <xsl:value-of select="po_id/@idref"/>
                </POID>
                <REMINVDATE/>
                <REMINVID/>
              </INVOICE>
            </xsl:for-each>
            <xsl:for-each select="//SabaObject[@type='com.saba.busobj.SabaInvoiceItem']">
              <xsl:variable name="ORDERITEMID">
                <xsl:value-of select="order_item_id/@idref"/>
              </xsl:variable>
              <xsl:for-each select="//SabaObject[@type='com.saba.busobj.SabaOrderItem']">
                <xsl:if test="$ORDERITEMID=@id">
                  <ITEM>
                    <ACCTID>
                      <xsl:value-of select="//account_id/@idref"/>
                    </ACCTID>
                    <TOTALCOST>
                      <xsl:value-of select="total_cost"/>
                    </TOTALCOST>
                    <DESCRIPTN>
                      <xsl:value-of select="description"/>
                    </DESCRIPTN>
                    <UNITCOST>
                      <xsl:value-of select="unit_cost"/>
                    </UNITCOST>
                    <ACTUALQTY>
                      <xsl:value-of select="actual_qty"/>
                    </ACTUALQTY>
                    <LINEID>
                  <xsl:value-of select="@id"/>
                </LINEID>
                <ATTRIBUTE1>
                    <xsl:value-of select="@id"/>
                </ATTRIBUTE1>
                    <xsl:variable name="STUDENTID">
                      <xsl:value-of select="student_id/@idref"/>
                    </xsl:variable>
                    <xsl:for-each select="//SabaObject[@id=$STUDENTID]">
                    <xsl:variable name="STUDENTNAME">
                      <xsl:value-of select="lname"/>,<xsl:value-of
select="fname"/>,Phone:<xsl:value-of select="workphone"/>
                    </xsl:variable>
                      <ATTRIBUTE2>
                  <xsl:value-of select="$STUDENTNAME"/>
                      </ATTRIBUTE2>
                    </xsl:for-each>
                  </ITEM>
                </xsl:if>
              </xsl:for-each>
            </xsl:for-each>
            <xsl:for-each select="//SabaObject[@type='com.saba.busobj.SabaInvoice']">
              <USERAREA>
                <OBJSTATUS>
```

```
            <xsl:value-of select="@status"/>
          </OBJSTATUS>
          <OBJTYPE>
            <xsl:value-of select="@type"/>
          </OBJTYPE>

    <AMOUNT_INCLUDES_TAX_FLAG>N</AMOUNT_INCLUDES_TAX_FLAG>
        </USERAREA>
      </xsl:for-each>
    </DATAAREA>
  </SYNC_INVOICE_001>
 </xsl:template>
</xsl:stylesheet>
```

The following is the equivalent Interchange Format document generated by the stylesheet transformation, an Invoice in OAG BOD format.

```
<SYNC_INVOICE_001>
<CNTROLAREA>
<BSR>
<VERB>SYNC</VERB>
<NOUN>INVOICE</NOUN>
<REVISION>001</REVISION>
</BSR>
<SENDER>
<LOGICALID/>
<COMPONENT/>
<TASK/>
<REFERENCEID/>
<CONFIRMATION/>
<LANGUAGE/>
<CODEPAGE/>
<AUTHID/>
</SENDER>
<DATETIME qualifier="CREATION">
<YEAR>1-10</YEAR>
<MONTH>20</MONTH>
<DAY>0-</DAY>
<HOUR/>
<MINUTE/>
<SECOND/>
<SUBSECOND/>
<TIMEZONE/>
</DATETIME>
</CNTROLAREA>
<DATAAREA>
<INVOICE>
<INVDATE>2000-11-10 19:17:40.000</INVDATE>
<CURRENCYID>http://spanuganti/interconnect/Saba/com.saba.interconn
ect.ObjectID@14966/34</CURRENCYID>
<INVNO>001000</INVNO>
<INVOICEID>invce000000000001000</INVOICEID>
<TOTALCHARGES>425.0</TOTALCHARGES>
<ACCTID>http://spanuganti/interconnect/Saba/com.saba.interconnect.
ObjectID@94902deb/206</ACCTID>
```

```
<CREATEDID>http://spanuganti/interconnect/Saba/com.saba.interconne
ct.ObjectID@170064/6</CREATEDID>
<UPDATEDON>2000-11-10 19:17:40.000</UPDATEDON>
<ORDERID/>
<BALANCE>425.0</BALANCE>
<AMTPAID>0.0</AMTPAID>
<OTHERCHARGES>0.0</OTHERCHARGES>
<STATUS>100</STATUS>
<FLAGS>0000000000</FLAGS>
<SPLIT>domin000000000000001</SPLIT>
<POID/>
<REMINVDATE/>
<REMINVID/>
</INVOICE>
<ITEM>
<ACCTID>http://spanuganti/interconnect/Saba/com.saba.interconnect.
ObjectID@94902deb/206</ACCTID>
<TOTALCOST>0.0</TOTALCOST>
<DESCRIPTN>Default   Default</DESCRIPTN>
<UNITCOST>0.0</UNITCOST>
<ACTUALQTY>1</ACTUALQTY>
<LINEID>ordit000000000001060</LINEID>
<ATTRIBUTE1>ordit000000000001060</ATTRIBUTE1>
</ITEM>
<ITEM>
<ACCTID>http://spanuganti/interconnect/Saba/com.saba.interconnect.
ObjectID@94902deb/206</ACCTID>
<TOTALCOST>425.0</TOTALCOST>
<DESCRIPTN>Inventory1</DESCRIPTN>
<UNITCOST>425.0</UNITCOST>
<ACTUALQTY>1</ACTUALQTY>
<LINEID>ordit000000000001061</LINEID>
<ATTRIBUTE1>ordit000000000001061</ATTRIBUTE1>
</ITEM>
<USERAREA>
<OBJSTATUS>new</OBJSTATUS>
<OBJTYPE>com.saba.busobj.SabaInvoice</OBJTYPE>
<AMOUNT_INCLUDES_TAX_FLAG>N</AMOUNT_INCLUDES_TAX_FLAG>
</USERAREA>
</DATAAREA>
</SYNC_INVOICE_001>
```

At step 7, the Monitor 1025 receives the Interchange Format document back from the Accessor 1035. At step 8, the Monitor 1025 instructs the Requestor 1020 to deliver the Invoice to the SAP system. At step 9, the Process Invoice document is actually delivered over the network to the SAP system. The Requestor 1020 reliably ensuring that the Invoice is actually delivered and received. At step 10, the Process Invoice document is inserted into the SAP

system as a new Invoice. Step 10 is performed by the SAP Importer. There are several possibilities for the implementation of the SAP Importer, depending on the level of functionality provided by SAP: (1) SAP supports the Interchange Document format directly, in which case this step is trivial, or (2) SAP supports a proprietary XML format, in which case a stylesheet can be used to transform the Invoice into SAP's proprietary format, or (3) SAP supports a proprietary API, which is used to read and process the XML document, either in its original format or after a stylesheet transformation into a more convenient format.

As another example, an employee record maintained in an external system is reflected in a SABA site. An administrator registers a callback event with an Interconnect enabled human resources (HR) system. A change in the HR system generates an event that is captured by the external system Monitor. The Monitor requests the HR data from the Accessor. The external system Accessor generates the updated HR record as an Interchange Document. The following is another example Interchange Format document, a Sync Personnel BOD:

```
<SYNC_EMPLOYEE_001>
<CNTROLAREA>
<BSR>
<VERB>SYNC</VERB>
<NOUN>EMPLOYEE</NOUN>
<REVISION>001</REVISION>
</BSR>
<SENDER>
<LOGICALID/>
<COMPONENT/>
<TASK/>
<REFERENCEID/>
<CONFIRMATION/>
<LANGUAGE/>
<CODEPAGE/>
<AUTHID/>
</SENDER>
<DATETIME qualifier="CREATION">
<YEAR/>
```

```
        <MONTH/>
        <DAY/>
        <HOUR/>
        <MINUTE/>
        <SECOND/>
        <SUBSECOND/>
        <TIMEZONE/>
        </DATETIME>
        </CNTROLAREA>
        <DATAAREA>
        <SYNC_EMPLOYEE>
        <EMPLOYEE>
        <NAME index="1">MR.</NAME>
        <NAME index="2">testfirst</NAME>
        <NAME index="3">testlast</NAME>
        <EMPLOYEEID>http://bnemazie/interconnect/Saba/com.saba.inter
connect.ObjectID@170179/6805</EMPLOYEEID>
        <EMPLOYEETYPE>Permanent</EMPLOYEETYPE>
        <SYNCIND/>
        <DUNSNUMBER/>
        <ADDRESS>
        <ADDRLINE index="1"/>
        <ADDRLINE index="2"/>
        <CITY/>
        <COUNTRY/>
        <POSTALCODE/>
        <STATEPROVN/>
        <TELEPHONE1/>
        <TELEPHONE2/>
        <FAX1/>
        <PARENTID/>
        <EMAIL/>
        </ADDRESS>
        <NAME2/>
        <CURRENCY/>
        <DESCRIPTN/>
        </EMPLOYEE>
        <USERAREA>
```

```
        <MNAME/>
        <TERRITORYID/>
        <COMPANYID/>
        <STARTEDON>2000-07-24 00:00:00.0</STARTEDON>
        <TERMINATEDON/>
        <LOCATIONID>http://bnemazie/interconnect/Saba/com.saba.inter
connect.ObjectID@cd92/6801</LOCATIONID>
        <RATE/>
        <SSNO>111-11-2222</SSNO>
        <GENDER>0</GENDER>
        <SHORTDESCRIPTN/>
        <JOBTYPEID/>
        <MANAGERID/>
        <QUOTA/>
        <UPDATEDON>provide</UPDATEDON>
        <UPDATEDBY>provide</UPDATEDBY>
        <MAXDISCOUNT/>
        <HOMEDOMAIN/>
        <USERNAME>1093-202</USERNAME>
        <FLAGS>0</FLAGS>
        <PASSWORD/>
        <STATUS>Full Time</STATUS>
        <LOCALEID/>
        <EMPLOYEENO>185</EMPLOYEENO>
        <SPLIT/>
        <CREATEDON>provide</CREATEDON>
        <OBJTYPE/>
        <OBJSTATUS>new</OBJSTATUS>
        <DESIREDJOBTYPEID/>
        </USERAREA>
        </SYNC_EMPLOYEE>
        </DATAAREA>
        </SYNC_EMPLOYEE_001>
```

The Monitor then receives the BOD from the Accessor and instructs the external system Requestor to deliver the personnel change to the SABA system.

The Requestor then delivers the Sync Personnel document over the network to the SABA system. The SABA Updater receives the Sync Personnel document. It uses an XSL stylesheet to transform the document into the canonical format used internally. The following is an example XSL personnel stylesheet:

```
<xsl:stylesheet
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<!--COPYRIGHT NOTICE Copyright (c) 1997-2000 Saba
Software Inc., 2400 Bridge
Parkway, Redwood Shores, California 94065-1166 USA. All
rights reserved.-->
<xsl:output indent="yes" method="xml" omit-xml-
declaration="no"/>
<xsl:template match="*|/">
<xsl:apply-templates/>
</xsl:template>
<xsl:template match="text()|@*">
<xsl:value-of select="."/>
</xsl:template>
<xsl:template match="SYNC_EMPLOYEE_001">
<xsl:for-each select="/">
<SabaObjectSerialization xmlns:dt="urn:w3-
org:xmldatatypes">
<SabaObject>
<xsl:attribute
name="type">com.saba.busobj.SabaEmployee</xsl:attribute>
<xsl:attribute name="status">
<xsl:value-of
select="//USERAREA/OBJSTATUS"/>
<xsl:if test="//USERAREA/OBJSTATUS=''"/>
</xsl:attribute>
<xsl:attribute name="id">
<xsl:value-of select="//EMPLOYEEID"/>
<xsl:if test="//EMPLOYEEID=''"/>
</xsl:attribute>
<title dt:type="string" dt:size="10">
<xsl:value-of select="//NAME[1]"/>
</title>
```

```
                        <fname dt:type="string" dt:size="25">
                           <xsl:value-of select="//NAME[2]"/>
                           <xsl:if test="//NAME[2]=''"/>
                        </fname>
                        <lname dt:type="string" dt:size="25">
                           <xsl:value-of select="//NAME[3]"/>
                           <xsl:if test="//NAME[3]=''"/>
                        </lname>
                        <mname dt:type="string" dt:size="25">
                           <xsl:value-of select="//USERAREA/MNAME"/>
                        </mname>
                        <homephone dt:type="string" dt:size="25">
                           <xsl:value-of select="//TELEPHONE1"/>
                        </homephone>
                        <workphone dt:type="string" dt:size="25">
                           <xsl:value-of select="//TELEPHONE2"/>
                        </workphone>
                        <fax dt:type="string" dt:size="25">
                           <xsl:value-of select="//FAX1"/>
                        </fax>
                        <created_on dt:type="string" updateFlag="No">
                           <xsl:attribute
name="provide">true</xsl:attribute>
                        </created_on>
                        <created_by dt:type="string" updateFlag="No">
                           <xsl:attribute
name="provide">true</xsl:attribute>
                        </created_by>
                        <updated_by dt:type="string">
                           <xsl:attribute
name="provide">true</xsl:attribute>
                        </updated_by>
                        <updated_on dt:type="dateTime">
                           <xsl:attribute
name="provide">true</xsl:attribute>
                        </updated_on>
                        <territory_id>
                           <xsl:attribute name="idref">
```

```
                                <xsl:value-of
select="//USERAREA/TERRITORYID"/>
                                </xsl:attribute>
                            </territory_id>
                            <custom0 dt:type="string">
                                <xsl:value-of
select="//USERAREA/CUSTOM0"/>
                            </custom0>
                            <custom1 dt:type="string">
                                <xsl:value-of
select="//USERAREA/CUSTOM1"/>
                            </custom1>
                            <custom2 dt:type="string">
                                <xsl:value-of
select="//USERAREA/CUSTOM2"/>
                            </custom2>
                            <custom3 dt:type="string">
                                <xsl:value-of
select="//USERAREA/CUSTOM3"/>
                            </custom3>
                            <custom4 dt:type="string">
                                <xsl:value-of
select="//USERAREA/CUSTOM4"/>
                            </custom4>
                            <company_id>
                                <xsl:attribute name="idref">
                                    <xsl:value-of
select="//USERAREA/COMPANYID"/>
                                    <xsl:if
test="//USERAREA/COMPANYID=''">bisut000000000000001</xsl:if>
                                </xsl:attribute>
                            </company_id>
                            <addr1 dt:type="string" dt:size="80">
                                <xsl:value-of select="//ADDRLINE[1]"/>
                            </addr1>
                            <addr2 dt:type="string" dt:size="80">
                                <xsl:value-of select="//ADDRLINE[2]"/>
                            </addr2>
```

```
<city dt:type="string" dt:size="50">
   <xsl:value-of select="//CITY"/>
</city>
<state dt:type="string" dt:size="50">
   <xsl:value-of
select="//ADDRESS/STATEPROVN"/>
</state>
<zip dt:type="string" dt:size="80">
   <xsl:value-of select="//POSTALCODE"/>
</zip>
<country dt:type="string" dt:size="80">
   <xsl:value-of select="//COUNTRY"/>
</country>
<email dt:type="string">
   <xsl:value-of select="//EMAIL"/>
</email>
<employee_no dt:type="string" updateFlag="No"
dt:size="80">
   <xsl:value-of select="//EMPLOYEENO"/>
   <xsl:if test="//EMPLOYEENO=''"/>
</employee_no>
<status dt:type="number">
   <xsl:value-of select="//USERAREA/STATUS"/>
   <xsl:if test="//USERAREA/STATUS=''">Full
Time</xsl:if>
</status>
<password dt:type="string" updateFlag="No">
   <xsl:value-of
select="//USERAREA/PASSWORD"/>
   <xsl:if
test="//USERAREA/PASSWORD=''">412ABF98CDF3EF99</xsl:if>
</password>
<username dt:type="string" updateFlag="No">
   <xsl:value-of
select="//USERAREA/USERNAME"/>
</username>
<manager_id>
   <xsl:attribute name="idref">
```

```
                                    <xsl:value-of
select="//USERAREA/MANAGERID"/>
                                 </xsl:attribute>
                              </manager_id>
                              <emp_type>
                                 <xsl:value-of select="//EMPLOYEETYPE"/>
                                 <xsl:if test="//EMPLOYEETYPE=''"/>
                              </emp_type>
                              <started_on dt:type="dateTime">
                                 <xsl:value-of
select="//USERAREA/STARTEDON"/>
                              </started_on>
                              <terminated_on dt:type="dateTime">
                                 <xsl:value-of
select="//USERAREA/TERMINATEDON"/>
                              </terminated_on>
                              <location_id>
                                 <xsl:attribute name="idref">
                                    <xsl:value-of
select="//USERAREA/LOCATIONID"/>
                                    <!-- Change value for default
location_id -->
                                    <xsl:if
test="//USERAREA/LOCATIONID=''">locat000000000001000</xsl:if>
                                 </xsl:attribute>
                              </location_id>
                              <max_discount dt:type="number">
                                 <xsl:value-of
select="//USERAREA/MAXDISCOUNT"/>
                                 <xsl:if
test="//USERAREA/MAXDISCOUNT=''">0</xsl:if>
                              </max_discount>
                              <split dt:type="string">
                                 <xsl:value-of select="//USERAREA/SPLIT"/>
                                 <xsl:if
test="//USERAREA/SPLIT=''">domin000000000000001</xsl:if>
                              </split>
                              <rate dt:type="number">
```

```
                        <xsl:value-of select="//USERAREA/RATE"/>
                        <xsl:if
test="//USERAREA/RATE=''">0</xsl:if>
                     </rate>
                     <quota dt:type="number">
                        <xsl:value-of select="//USERAREA/QUOTA"/>
                        <xsl:if
test="//USERAREA/QUOTA=''">0</xsl:if>
                     </quota>
                     <jobtype_id>
                        <xsl:attribute name="idref">
                           <xsl:value-of
select="//USERAREA/JOBTYPEID"/>
                        </xsl:attribute>
                     </jobtype_id>
                     <ss_no dt:type="string">
                        <xsl:value-of select="//USERAREA/SSNO"/>
                        <xsl:if test="//USERAREA/SSNO=''"/>
                     </ss_no>
                     <gender dt:type="number">
                        <xsl:value-of select="//USERAREA/GENDER"/>
                        <xsl:if test="//USERAREA/GENDER=''"/>
                     </gender>
                     <home_domain>
                        <xsl:attribute name="idref">
                           <xsl:value-of
select="//USERAREA/HOMEDOMAIN"/>
                           <xsl:if
test="//USERAREA/HOMEDOMAIN=''">domin000000000000001</xsl:if>
                        </xsl:attribute>
                     </home_domain>
                     <desired_job_type_id>
                        <xsl:attribute name="idref">
                           <xsl:value-of
select="//USERAREA/DESIREDJOBTYPEID"/>
                        </xsl:attribute>
                     </desired_job_type_id>
                     <locale_id>
```

```
                        <xsl:attribute name="idref">
                           <xsl:value-of
select="//USERAREA/LOCALEID"/>
                           <xsl:if
test="//USERAREA/LOCALEID=''">local000000000000001</xsl:if>
                        </xsl:attribute>
                     </locale_id>
                     <flags dt:type="string">
                        <xsl:value-of select="//USERAREA/FLAGS"/>
                        <xsl:if
test="//USERAREA/FLAGS=''">0000000000</xsl:if>
                     </flags>
                     <timezone_id>
                        <xsl:attribute name="idref">
                           <xsl:value-of select="//TIMEZONE"/>
                           <!-- Change value for default
timezone_id -->
                           <xsl:if
test="//TIMEZONE=''">tzone000000000000008</xsl:if>
                        </xsl:attribute>
                     </timezone_id>
                  </SabaObject>
               </SabaObjectSerialization>
            </xsl:for-each>
         </xsl:template>
      </xsl:stylesheet>
```

The following is the equivalent Local Format document, a generated Saba Person in Saba Canonical Format:

```
      <SabaObjectSerialization xmlns:dt="urn:w3-org:xmldatatypes">
      <SabaObject type="com.saba.busobj.SabaEmployee"
status="existing"
id="http://bnemazie/interconnect/Saba/com.saba.interconnect.Object
ID@170179/6805">
      <title dt:type="string" dt:size="10">MR.</title>
      <fname dt:type="string" dt:size="25">testfirst</fname>
```

```
        <lname dt:type="string" dt:size="25">testlast</lname>
        <mname dt:type="string" dt:size="25"/>
        <homephone dt:type="string" dt:size="25">972 580
7645</homephone>
        <workphone dt:type="string" dt:size="25"/>
        <fax dt:type="string" dt:size="25"/>
        <updated_by dt:type="string" provide="true"/>
        <updated_on dt:type="dateTime" provide="true"/>
        <territory_id idref=""/>
        <custom0 dt:type="string"/>
        <custom1 dt:type="string"/>
        <custom2 dt:type="string"/>
        <custom3 dt:type="string"/>
        <custom4 dt:type="string"/>
        <company_id idref="bisut000000000000001"/>
        <addr1 dt:type="string" dt:size="80">1213 addr1 1234</addr1>
        <addr2 dt:type="string" dt:size="80"/>
        <city dt:type="string" dt:size="50">Irving</city>
        <state dt:type="string" dt:size="50">TX</state>
        <zip dt:type="string" dt:size="80">75038</zip>
        <country dt:type="string" dt:size="80">US</country>
        <email dt:type="string"/>
        <employee_no dt:type="string" dt:size="80">185</employee_no>
        <status dt:type="number">Full Time</status>
        <password dt:type="string">412ABF98CDF3EF99</password>
        <username dt:type="string">1093-202</username>
        <manager_id idref=""/>
        <emp_type>Permanent</emp_type>
        <started_on dt:type="dateTime">2000-07-24
00:00:00.0</started_on>
        <terminated_on dt:type="dateTime"/>
        <location_id
idref="http://bnemazie/interconnect/Saba/com.saba.interconnect.Obj
ectID@cd92/6801"/>
        <max_discount dt:type="number">0</max_discount>
        <split dt:type="string">domin000000000000001</split>
        <rate dt:type="number">0</rate>
        <quota dt:type="number">0</quota>
```

```
<jobtype_id name="idref"/>
<ss_no dt:type="string">111-11-2222</ss_no>
<gender dt:type="number">0</gender>
<home_domain idref="domin000000000000001"/>
<desired_job_type_id idref=""/>
<locale_id idref="local000000000000001"/>
<flags dt:type="string">0</flags>
<timezone_id idref="tzone000000000000008"/>
</SabaObject>
</SabaObjectSerialization>
```

A SabaEmployee object is instantiated based on the canonical XML document. This object is then saved, committing any changes to the database.

The set of interconnect components is extensible so additional functionality can be added over time. Adding a Searcher component allows a site to be "exchange enabled" – able to share catalog (or other) information with other sites. In this way users can get results from searches that combine remote catalog offerings with local catalog offerings. Adding a Purchaser component makes a site "eCommerce enabled" – able to offer products for sale via an automated interface. This enables learners who choose classes from a catalog that has been shared on SabaNet to purchase them via SabaNet. A Versioner component could offer the ability to automatically upgrade to the latest version of the software or to automatically purchase a license extension via a Licensor component.

As described above the DeliveryService is a key component of the Interconnect Backbone. Interconnect messages follow an persistent asynchronous protocol. Messages are sent and received with a message payload. Message payloads are opaque to the DeliveryService, any object may be sent as a message payload. A message recipient may reply to a message by constructing a reply message from the original message and sending that reply as a separate asynchronous message.

Message senders and recipients are responsible for synchronizing their own messages. There are message ID fields in the Message that may be used for this purpose. A Message contains (1) The sender's InterconnectAddress (2) The recipient's InterconnectAddress (3) The sender's credentials (4) A messageID (5)

A replyID (6) The message payload (an Object). Message senders and recipients have an InterconnectAddress. This Address is managed by the DeliveryService and contains (1) An Inbox identifier (InboxID) assigned by the local DeliveryService (2) A String in URI format identifying the service (mServiceURI), (3) An Object identifying the associated User (mUser).

The InboxID is used by a DeliveryService for local message routing. The URI identifies the specific software component and is used to determine whether the InterconnectAddress is local or remote. To send a message, an Interconnect client must: (1) construct a Message for the given sender and recipient, (2) add the message payload to the message, (3) set the message ID or the reply ID if needed, (4) send the message using the DeliveryService's IPostman interface. If the message is local it will be delivered using the InboxID. If it is remote it will be forwarded to the appropriate remote DeliveryService for delivery at that location.

In order to use the DeliveryService, a connect must first be made. Upon connection the DeliveryService assigns an InboxID that is used internally for message routing and synchronization. This InboxID is used in subsequent calls to the DeliveryService.

Once connected, messages may be sent or received from the DeliveryService. There are two ways messages can be delivered depending upon how the recipient registers. The recipient may Poll for messages using IPostman.getMessage() or handle incoming messages by implementing IRecipient.recieveMessage(). The IPostman.connect() method has an optional IRecipient parameter. If a valid IRecipient is passed, incoming messages will be delivered using that interface. In this case, behind the scenes, an InboxAssistant is created in a separate thread to watch the Inbox on behalf of the recipient. When a message is sent using IPostman.sendMessage() the DeliveryService is responsible for making sure that the message gets delivered to the appropriate Inbox. If it cannot it must report or log an error.

In the simple case where a message recipient is in the same installation as the sender, the DeliveryService will put the message in the recipient's Inbox and be done with it. The message will stay there until the recipient or the

InboxAsistant takes it out. When finished using the service, an Interconnect client may disconnect from or release the Inbox. Disconnecting tells the DeliveryService to maintain messages as the recipient intends to reconnect at some later time. Releasing frees all DeliveryService resources associated with the Inbox.

When the DeliveryService determines that message is destined for a recipient in another Interconnect location, the local DeliveryService must forward the message to its peer DeliveryService at that location. The service identifier in the message's recipient address is used to determine whether the recipient is local or remote. This identifier is a URI with the Host name (as returned by InetAddress.getLocalHost()).getHostName()) and Interconnect service name. For example, a service named "SabaAccessor" running on Saba host "flamenco" would have an URI of the form "rmi://flamenco.saba.com/SabaWeb/Saba/Accessor".

The ServiceManager will look at the serviceURI and determine whether the service in remote or local, if it is remote it will resolve the address with it's remote peer.

Key to the design of the Interconnect is the notion of pluggable transport protocols. To accommodate this, the Delivery Service has 2 components (1) Delivery Service (2) Persistent Message Manager. The Delivery Service writes messages to outbound queues (if the message needs to be delivered to an external system), the Persistent Message Manager polls out bound queues to deliver the message to the host the message is intended for. The persistent Message Manager has the uses pluggable transport protocol. For implementing a protocol using RMI a class needs to be written implementing IPMTransport. The Persistent Message Manager (PMM) acts as the listener for receiving messages. Messages received are put into inbound queues, the Delivery Service delivers messages from the inbound queues to the Subscribers.

The rationale behind this separation is to allow for the Interconnect DeliveryService/PMM to be deployed across a wide variety of communication protocols. Supporting a new protocol requires building a delivery transport that wraps that protocol. The protocol wrappers are implemented as peers, and initiate

and accept connections, send and receive messages, terminate gracefully, etc. For example, the following steps would be performed to build a TCP/IP socket Interconnect Transport:

1. Implement a interconnect listener/accepter
2. Implement a client connection initiator
3. marshal and write interconnect messages onto a socket
4. read and unmarshal interconnect messages from a socket
5. implement the IPMTransport interface

A discussion of mapping Ids from one system to another using the POID concept follows. When the Accessor receives a request to export an object to a stream, it is passed a user object and a platform ID (POID). In this case the POID is an ID associated with the local object in this system. Generally this ID will be acquired from another exported document or as a result of a Monitor event however, some initial mappings may need to be provided to bootstrap the system.

Given the POID, the Accessor looks up the local ID and the document type in the Mapper. It is an error if there is no associated local object. The Accessor then uses the document type to look up the appropriate stylesheet, transformer and XMLHelper to use during the accessing and transformation steps.

Using the AccessorReader for the configured system, the local object is extracted into a stream in a system specific XML format. The XML stream, the stylesheet and an output stream are then passed to the transformer that writes the transformed XML to the new stream. The transformed stream is then returned.

This is in the simple case where the XML to export contains no external references to objects in the source system which are not contained in the generated XML. In the more complicated case, the XML stream is not fully self contained, i.e. it contains references to objects that are not part of the XML stream. XML however may contain the local Object Id of this Object, this Id is meaning less outside this system. This Id needs to be replaced with its POID.

The Accessor service needs to attempt to insure that all unresolved references in the outbound XML document are represented in the form of a POID.

During export, the Accessor must find or create a POID for each reference encountered and fix up those object references in the XML stream. The Accessor will use the Mapper to determine if the referenced object has an associated POID. If a POID does not exist, one will be created and added to the Mapper's tables.

Step by step on the Accessor side:

1. The Accessor requests a document be exported by invoking the Accessor method:

   Reader IAccessor.getObjectReader(UserObject user, POID poid)

2. The Accessor looks up the local object ID from the Mapper:

   LocalObjectID Mapper.getLocalID(POID platformID)

   If there is no local ID an exception is raised.

3. The Accessor looks up the document type from the Mapper:

   String Mapper. getDocumentType (POID platformID)

   If there is no document type, a default is used for the configured AccessorReader.

4. The Accessor looks up the stylesheet, IXMLHelper and ITransformer using the docType.

5. The Accessor requests the object in XML format from the AccessorReader:

   Reader IAccessorReader.extractObjectReader(LocalObjectID localID, IXMLHelper helper)

6. The Accessor fixes up ID references in the XML stream. It scans the stream looking for foreign POIDs.

7. When a reference ID is encountered by the Accessor, it resolves it to the POID using the Mapper. If no POID exists one is created. The POID is written to the XML stream.

8. An output stream is created and the document is transformed:

void ITransformer.transform(String stylesheet, Reader in, Writer out)

When the Importer receives a request to import an object from a stream, it is passed the stream, a user object, the document type and a platform ID (POID). This POID is a foreign ID, created when the document is exported from the source system.

The XML stream, a stylesheet and an output stream are passed to the transformer and a new XML stream is produced. This new stream is passed to a platform specific object that inserts it into the system. On insert, a local object ID is created by the system and returned.

When the local ID is returned to the Importer, the Importer asks the Mapper to map the foreign POID to the Local Object. The POID is then returned to the requestor in the import status reply.

This is in the simple case where the XML to import contains no external references to objects in the source system which are not resolved in the XML.

In the more complicated case, the XML document not fully self contained. The document to import contains references to objects that are not part of the XML document. The import service attempts to resolve these references to insure the referential integrity of the object being imported. During the transformation phase, the Importer must resolve the foreign references to local objects and fix up those object references in the XML stream.

The specified object may have already been imported in which case there will be an entry in the local Mapper's foreign POID map. The Importer asks the mapper to resolve the POID to a local object. If this object has been mapped, a string representation of the Object ID is used to replace the foreign POID in the XML document.

In the case where the object has not been previously imported the importer has two choices. Either it can fail and report an error, or it can attempt to pull the

object from the foreign system. It is reasonable to make this a configurable option and perhaps only support error reporting in the initial release.

Step by step Id mapping on Import:

1. The Subscriber requests a document be imported by invoking the IImporter method:

ImportStatus IImporter.importObjectFromStream(POID poid, UserObject user, Reader stream, String docType)

2. The Importer looks up the stylesheet, IXMLHelper and ITransformer using the docType.

3. An output stream is created and the document is transformed:

void ITransformer.transform(String stylesheet, Reader in, Writer out)

4. The Importer fixes up foreign ID references in the XML stream. It scans the stream looking for foreign POIDs.

5. When a foreign ID is encountered by the Importer, it resolves it to the local ID using its Mapper. The local ID is written to the XML stream.

LocalObjectID Mapper.resolveForeignObject(POID foreignID)

6. The fixed-up XML stream is passed to the ImporterWriter to insert into the system.

LocalObjectID insertObjectFromStream(Reader in, IXMLHelper helper)

7. Map the new local ID to the original foreign POID passed with the import request.

void Mapper.mapForeignObject(POID foreignID, LocalObjectID localID)

So far the discussion has been around the Interconnect/Connector framework. The following discusses Connector Specific plug ins, and defines the specific components for each connector. Taking Saba Connector as an example:

a. SabaChangeManager – This class extends the Change Manager, starts a thread that polls the database for changes. Once a change is detected the change is passed over to the Monitor for further processing. This class has the specific logic to poll Saba database.
b. SabaImporterWriter – This class extends the ImporterWriter and has the logic to import Objects in Local format (SCF)into Saba system.
c. SabaAccessorReader – This class extends the AccessorReader and has the specific logic to retrieve objects from Saba system in local format.

Every new connector has to implement these 3 classes to work with application connecting. Extending this we have sapChangeManager, sapImporterWriter and sapAccessorReader.

INFORMATION SERVER

The present invention relates to a novel information distributor method and apparatus. The present invention can provide services for consolidating, analyzing, and delivering information that is personalized, relevant, and needed.

It employs metadata-based profiles to match information with users. User profiles may include skill competencies and gaps, roles and responsibilities, interests and career goals.

The Platform services provides the interface and infrastructure for building agents that work in concert to decide what information is delivered, when it is delivered, and how it is delivered.

The platform services integrate with the Platform Interconnect Server to work across different networks and disparate information systems. This allows

users to receive information from a variety of sources and locations via a single, consistent interface.

The present invention uses an Information Distributor Developer's Kit (IDK) to be used by software application developers of ordinary skill in the art.

The platform of the present invention identifies and fills information gaps across the corporate value chain. IDK provides the infrastructure and core functionality to find and deliver relevant and targeted information. In an embodiment, the IDK enables more sophisticated querying and matching functionality than in the prior art and serves as the technology underpinnings for a stand-alone Enterprise Information Portal (EIP) solution.

For more information on RDF, refer to the W3C home page, incorporated by reference in its entirety, at the URL www.w3.org/RDF/ and formal specification located at URL www.w3.org/TR/REC-rdf-syntax/.

The above sources of information are incorporated by reference in their entireties.

Figure 11 shows a structural overview of an IDK 1100 of the present invention. IDK 1100 is associated with a language 1102, such as RDF, for representing web metadata, a language for querying web metadata, and a set of APIs 1104 for defining information services based on what data is used, when and how a match is performed, and what is done with the results.

Figure 12 shows a functional overview of an Information Distributor 1201 of the present invention. IDK 1100 can annotate and match broad resources 1200, support diverse sources, conditions, and delivery options 1202, provide an easy migration path 1204, and leverage open standards 1206.

In an embodiment of the invention, Information Distributor 1201 provides a flexible mechanism for annotating and matching web resources 1200. Information Distributor 1201 can locate and deliver a wide variety of resources, from web pages to Business Objects. Information Distributor 1201 also supports a wide variety of descriptive information required by business applications, from standard web metadata to catalog information to skills and competencies.

Information Distributor 1201 also supports a broad variety of information sources, match conditions, and delivery mechanisms 1202. Information Distributor 1201 generates matches under a variety of circumstances and supports a variety of options for delivering match results.

Information Distributor 1201 provides an easy migration path 1204. A software developer of ordinary skill in the art can write queries using a combination of Java code and SQL. IDK provides equivalent functionality using a higher-level languages for representing and querying data and simpler programming APIs. Information Distributor 1201 also leverages open standards 1206 by supporting industry standards such as RDF and XML. Support for industry standards helps ensure the availability of third-party tools that interoperate with IDK and increases the set of data and information on which IDK can act.

In an embodiment of the invention, Information Distributor 1201 can determine if a new software developer has just joined a new project. If one of the skills required for the new software developer's new assignment is knowledge of XML, then upon joining the project, Information Distributor 1201 automatically send an email to the new software developer containing information about the company's standard "Introduction to XML" course.

In an embodiment of the invention, Information Distributor 1201 can keep a development manager informed about the status of the other development groups in his division. As part of his custom home page provided by the corporate portal, he can view a list of the most recent updates submitted by each development manager, and call up each report in his web browser.

In an embodiment of the invention, Information Distributor 1201 can detect when an affiliated training provider has made available a new advanced class in Java. Information Distributor 1201 sends email to all advanced and expert Java programmers in the company announcing the availability of this class.

In an embodiment of the invention, Information Distributor 1201 can detect when the HR department institutes a new approval practice for all new hires. Information Distributor 1201 assures all hiring managers in the company

receive a new entry in the Corporate Information channel that explains the policy change.

If an updated price list for a region is generated, Information Distributor 1201 sends an email containing the new price information to all dealers in that region.

If an employee has a change in his family status, such as if the employee has a baby, the next time the employee views the HR department's benefits page in his web browser, the Information Distributor assures customized plan and deductible information appears that is appropriate for his new family status.

Referring again to Figure 11, in an embodiment, the Information Distributor adopts a new standard for web metadata and its definition of a high-level language 1102 for querying this metadata.

Metadata is structured information about information, and is used to identify, categorize, and locate resources of interest. Resource Description Format (RDF) is a new, XML-based standard for associating arbitrary metadata with any web resource. It can be used to describe resources ranging from a course catalog on the WWW to a business object representing a client.

In an embodiment a language used to query web metadata 1102 may be RDF Query Language (RQL), an XML-based query language for writing queries against RDF data. It can represent both simple and complex queries, and can also accommodate metadata matching, where a metadata description can be part of the query. For example, this allows a particular employee's complete skills gap – expressed as an RDF description – to be used in a query to locate classes that fill the gap.

Figure 13 shows an exemplary view of APIs 1104 associated with the Information Distributor. In an embodiment, the Information Distributor partitions information matching and delivery issues into three areas, each addressed by a distinct type of agent, Import Agents 1300, Match Agents 1302, and Delivery Agents 1304. The combination of Import Agent 1300, Match Agents 1302, and Delivery Agents 1304 is a novel combination of the present invention.

Import Agents 1300 create and import the RDF descriptions used by IDK. Import Agents 1300 can generate metadata from a variety of sources, from existing web pages and business objects to content management systems to enterprise applications.

Match Agents 1302 determine what matches and queries occur under what conditions. Match Agents 1302 can be triggered by a request to a web or application server, by specific events, or on a regularly scheduled basis. A Match Agent 1302 also specifies the RQL and any input metadata to use as the metadata query.

Delivery Agents 1304 dispatch the results of a query or match. In an embodiment, Delivery Agents 1304 integrate with a variety of delivery mechanisms, from web page generation and XML datagrams to email and event messaging systems.

In an embodiment of the invention, Figure 14 shows an exemplary view of using Information Distributor or IDK 1100. A software developer of ordinary skill in the art can use IDK to query objects 1400 or to implement custom delivery service 1402. In an embodiment, Query Objects 1400 may be used similarly to today's finder methods, that is, a high-level mechanism to query SABA business objects, but using and requiring knowledge of RDF and RQL.

Figure 15 shows an exemplary overview of Query Objects 1400. The invention, through a user associated with the invention, such as but not limited to a software developer of ordinary skill in the art, defines RDF Metadata Mappings 1500 for the objects and metadata of interest. Then, the invention Authors An Import Agent 1502 to capture this metadata. The invention may then Author An RQL Document 1504 to query this metadata and author a Match Agent to Perform the Query 1506 and a Delivery Agent to act on the query results.

Figure 16 shows an exemplary overview of Implement Custom Delivery Service 1402. The invention, through a user, such as but not limited to a software developer of ordinary skill in the art, may use the invention's IDK to novelly Implement a Custom Information Delivery Service 1402, using RDF, RQL, and the full IDK interface. In an embodiment, the invention Defines RDF Metadata

Mappings 1600 for the objects and metadata of interest. The invention Authors An Import Agent 1601 to capture this metadata. The system and method of the present invention then Authors An RQL Document 1602 to query this metadata. The invention then Authors a Match Agent 1604 to perform the query. and Authors a Delivery Agent 1606 to dispatch the query results. The invention then Integrates All Agents 1608, including the import agent, the match agent, and the delivery agent, into the existing system.

In an embodiment of the invention, Information Distributor (IDK) is a Software Development Kit delivered as part of Platform 4.0. It provides the infrastructure and basic functionality needed to build and customize the Enterprise Information Portal.

IDK provides the infrastructure and services to perform metadata-based queries. Unlike traditional text-based search engines, in an embodiment the IDK operates solely on descriptive data about resources, rather than the resources themselves.

In an exemplary embodiment of the invention, referring again to Figure 13, IDK defines interfaces for metadata generation (Importers or Import Agents 1300) and matching (Resolvers or Match Agents 1302) and for delivering query results (Dispatchers or Delivery Agents 1304). Combinations of these three services allow the Information Distributor to interoperate with a variety of enterprise systems and to service queries in a broad range of application domains.

In an embodiment, a portal server may be delivered using IDK.

Import Agents are responsible for consolidating a variety of information sources. Importers integrate with various external systems, analyze the descriptive data about specific resources in the system, and import this data into a custom RDF database. Exemplary information sources include internal email systems and Intranets, SABA EMS, ERP systems, and the World Wide Web.

Common tasks supported by Import Agents include:

- Executing batch imports
- Scheduling imports at regular intervals
- Analyzing and translating metadata formats

- Specifying a target database
- Integrating with SABA Interconnect

Match Agents are responsible for matching between information resources and user profiles. Match Agents execute at regular intervals or in response to specific requests. They perform intelligent comparisons between metadata descriptions of imported resources and user profiles. These comparisons return a set of information resources as the match result.

Because they act on detailed user profiles, Match Agents can function as personal agents, identifying those resources most relevant to a user's job, interests, or objectives. For example, they can determine that a user requires knowledge of a specific technology for a new job assignment, and deliver suggestions for classes covering that technology.

Because they match against categorized metadata, Resolvers can return more accurate and meaningful results than is possible with traditional text-based searches. For example, Match Agents can return only documents that have been updated within the last week. Or they can distinguish between articles about an individual and articles written by the individual.

Delivery Agents are responsible for delivering the results of a match to the correct recipients in the appropriate fashion. Delivery Agents integrate with various delivery mechanisms, delivering either pointers to the match results or the actual information itself. Typical delivery vehicles include e-mail, web servers, and enterprise portals.

Common tasks supported by Delivery Agents include:

- Delivering results immediately upon availability
- Delivering results at delayed or batched intervals
- Integrating with SABA Interconnect

In an embodiment, the final system and method of the present invention may be capable of scaling to handle enterprise-wide document databases. An initial prototype that may be delivered is capable of demonstrating the proof-of-concept without exhibiting the scalability of the final system.

The IDK provides a flexible mechanism for describing and comparing a wide variety of resources. The actual data being compared may vary widely among applications, ranging from competencies and skills for gap analysis to document summaries and reviews for web content. Yet the actual operations involved in determining a match tend towards a small set, text and numeric comparisons and basic Boolean logic. Thus, the IDK needs to casts a broad variety of properties into a consistent format for purposes of comparison.

In an embodiment, the invention employs the Resource Description Format (RDF), the World Wide Web Consortium's standard for web metadata. It meets the above requirements because it is designed to support a wide range of different applications, expressing them all in a consistent attribute property/value format. The format also allows the definition of standard vocabularies for specific application domains, and the mixing and matching of these vocabularies to describe a resource. The format has a web-centric design, employing URLs to describe any form of web resource and XML to serialize its data graphs and is seeing slow but steady adoption in a variety of domains, from electronic documents and on-line learning to news stories and business cards.

By choosing RDF as the Information Distributor's standard metadata format, the invention makes it easy and efficient for customers to work with the system because they can turn to external sources for training and documentation, can use third party tools for defining their metadata, and are more likely to already have or be able to find developers familiar with RDF. Furthermore, as RDF is used for more domains, the Information Distributor can be applied to an ever-increasing amount of content.

RDF is essentially a model for representing attribute/value pairs as a directed labeled graph. It consists of statements that pair a web resource (anything identified by a URL) with a property and a value. At its core, IDK provides a flexible mechanism for comparing these attribute/value pairs and taking action upon the comparison results.

The Match Agent operates by comparing one RDF description to the full set of RDF descriptions in a specified database. Because of the variety and

flexibility of RDF descriptions, additional instructions are required to specify how the match is performed. This is the function of the match template.

Match templates specify certain fields as belonging to a target RDF file. In an embodiment, the target is a file that is provided along with the match template to customize the search, for example, to perform a predefined search against a specific individual's description. Match templates may also be written to perform a fixed search, in which case there is no target RDF file. Merging a match template with a target RDF file produces an RDF query.

Match templates can specify the following aspects of a query:

- The specific properties to be compared.
- The comparison operation (=, !=, <, >)
- Boolean operators (AND, OR, NOT)
- A set of comparison functions, including:
    - like (text matching)
    - latest (most recent date)
- container operation: contains, first, etc.

In an embodiment, match templates are:

- easy to create and edit by hand
- conducive to creation by an authoring tool
- easy to parse

In an embodiment, the complete syntax and specification used by match templates is defined by the RDF Query Language Specification, described below.

RDF-based Match Templates are unique and never before contemplated by the prior art. The combination of a match template and a target RDF file can produce an RDF Query. In an embodiment of the invention, the core of the Information Distributor is a RDF Query engine that performs a query on one or more RDF databases, then returns a set of resources that satisfy the query.

In an embodiment of the invention, a client may use the Information Distributor SDK by performing the following exemplary method steps:

1. Write an Import Agent that implements the `ImportAgent` interface and employs the `MR.importRDF()` method.
2. Write a Match Agent that implements the `MatchAgent` interface and employs the `MR.match()` method.
3. Write a Delivery Agent that implements the `DeliveryAgent` interface.
4. Create a new instance of an MR (Metadata Repository).
5. Write code to create specific instances of the above agents and set them into motion.

In an embodiment of the invention, an ImportAgent is responsible for delivering metadata in RDF format to a Metadata Repository. Specific Import Agents may interface with a particular source of metadata, translate that metadata into RDF, and use the MR.importRDF() method to import that RDF. Import Agents may register with the Event Manager to perform imports in response to particular events. In an embodiment, the Import Agent has the sole responsibility for performing the metadata translation. In an embodiment of the invention, the invention provides utility routines that assist with translating various common metadata formats or serve to automatically generate metadata. In an embodiment, the invention provides additional utility functions for interfacing with the Event Manager or scheduling batch imports.

In an embodiment of the invention, a MatchAgent is responsible for performing a metadata match. Specific Match Agents may create a Match Descriptor and pass it to a specific MR to perform a match. Match Agents may perform matches in response to particular events. In an embodiment of the invention, distributed queries may be performed across multiple MR.

Match Agents may employ a utility class called MatchDescriptor that captures all information needed for a metadata query or match template.

This class is defined as follows:

```
public class MatchDescriptor
{
```

```
        /** MatchDescriptor constructor.
         *
         * @param aTemplate Contents of a match template.
         * @param aTarget URI of a target RDF file.  May be NULL if the
match
         *        template describes a fixed search.
         * @param aHandler MatchHandler to operate on the match results.
         */
        public MatchDescriptor(String aTemplate, String aTarget,
MatchHandler aHandler)


    } /* MatchDescriptor */
```

In an embodiment of the invention, a Delivery Agent is responsible for delivering the result of a metadata match. Delivery Agents implement the following Java interface:

```
        public interface DeliveryAgent
        {

            /** Deliver the results of a match.
             * @param mrs A MatchResultSet containing the match
    results.
             * @exception DeliveryException Thrown when
    delivery fails.
             */
            public void deliver(MatchResultSet mrs) throws
    DeliveryException;

        } /* DeliveryAgent */
```

Delivery Agents use a utility class called MatchResultSet that contains the result of a metadata match. A MatchResultSet contains a Vector of RDFResource objects, a class containing a URI for each resource returned by a metadata match, as well as additional, optional properties. The MatchResultSet class is defined as follows:

```
public class MatchResultSet
{
    /**
     * Set the results.
     * @param theResults Vector of RDFDescription objects.
     */
    public void setResults(Vector theResults)

    /**
     * Return an Enumeration of match results.
     * @return Enumeration of RDFDescription objects
     */
    public Enumeration getResults()
```

In an embodiment of the invention, the contents of the MatchResultSet may be serialized as a simple XML document. One RDF Description element may be associated with each result. Using RDF permits the invention to deliver additional properties that may be useful to the consumer of the MatchResultSet, such as properties taken from the source RDF Description or additional properties returned by the Match Engine.

The following is pseudocode for a sample XML result:

```
*
* <resultset>
*   <Description about =
"http://sabainet/devo/status/sb11_12_99.html">
*     <dc:Title>Weekly Status of Project Sweet Baboo</dc:Title>
*   </Description>
*   <Description
about="http://sabainet/devo/status/lp11_08_99.html">
*     <dc:Title>Weekly Status of Project Beethoven</dc:Title>
```

```
*   </Description>
* </resultset>
*
```

In an embodiment of the invention, a MR (Metadata Repository) is an interface that any Metadata Repository must implement.

The following is the interface for a MR:

```
public interface MR
{

    /* The import methods are used to insert RDF
metadata into the MR. */

    /** Import an RDF document specified in a URI.
     * @param uri URI to the RDF file.
     * @exception ImportException Thrown when import
fails.
     */
    public void importRDF(String uri) throws
ImportException;

    /** Import an RDF document specified in a Reader.
     *
     * The "key" parameter serves as a unique
identifier;
     * when RDF is re-imported with the same key value,
it replaces the previous
     * import.  The "key" value is most typically the
URI.
     *
     * @param r Reader containing RDF text.
     * @param key Unique identifier for this RDF
source.
     * @exception ImportException Thrown when import
fails.
```

```
         */
        public void importRDF(Reader r, String key) throws
ImportException;

        /** Perform a metadata match.  This involves the
following steps:
         *
         * <ol>
         * <li>Extracting the contents of the
MatchDescriptor
         * <li>Generating a MatchResultSet
         * <li>Passing the MatchResultSet to the
MatchHandler contained
         * in the MatchDescriptor
         * </ol>
         *
         * @param md MatchDescriptor fully describing the
match to perform.
         * @exception MatchException Thrown when match
fails.
         */
        public abstract void match(MatchDescriptor md)
throws MatchException;

        /**
         * Retrieve a named property of a specific
resource.  Returns null if
         * the specified property does not exist.
         *
         * @param resource URI of resource.
         * @param namespace URI of namespace; null if no
namespace is specified.
         * @param property Property name.
         * @return Property value.
         */
        public String getProperty(String resource, String
namespace, String property) throws MatchException;
```

```
} /* MR */

"
```

In an embodiment of the invention, RDF Query Language (RQL) is an easy-to-learn, easy-to-author language for querying collections of RDF documents. It is designed to support the full functionality required by Information Distributor.

RQL is an XML application. An RQL document may consist of a single Select element containing a single Condition. A condition may be either a direct operation on a single property, or a Boolean grouping operation, which can in turn contain further Conditions. RQL can define a number of built-in comparison operations; it also allows comparisons against variables extracted from an accompanying target RDF file.

Each Element is described in detail below.

RDFQuery

RDFQuery is the root element of an RQL document. It must contain a single Select element.

Container

A container is a grouping property value. Containers can be Bags, unordered lists of resources or literals, Sequences, ordered lists of resources of literals, or Alternatives, distinct choices.

Literal

A literal is a property value that is a simple string (including possibly XML markup) or other primitive datatype.

Property

A property is a specific characteristic or attribute used to describe a resource. The RDF model may contain Statements, which are a named property and value assigned to a specific resource.

Resource

A resource may be anything described by an RDF expression. A resource is identified by a URI.

Select

The **Select** element defines the properties that are returned by an RDF Query. The result of an RDF Query is itself an RDF document; it is the set of RDF Description elements that satisfy the query. By default, only the Resource URI is returned (as an about, aboutEach, or aboutEachPrefix attribute of the Description element). The **properties** attribute is used to define additional properties to be returned. It is a space-separated list of all property names to be returned. The initial implementation only allows literal, first-level property values to be returned; that is, containers, nested properties, and resources are not supported.

Within the Information Distributor, the returned RDF elements are wrapped in a MatchResultSet object for convenient manipulation from Java.

Condition

The **Condition** element defines a condition that RDF Descriptions must satisfy to be returned. Conditions are either simple, in which case they specify a Property/Value/Operation triple, or complex, in which case they contain one of the boolean operators. The simple Conditions simply obtain a property and compare it to the value using the specified operation. Operations are defined for literal properties and container properties.

A Property/Value/Operation triple can also contain a nested Condition; this allows querying against reified statements, or statements about statements. Refer to Query 11 for an example.

And, Or, Not

The **Boolean** operators perform logical operations on one or more conditions. **Not** negates the value of a single conditions, while **And** and **Or** perform logical operations on two or more conditions.

Because many RQL operations operate on containers, there is an "applies" attribute that determines the behavior of grouping operators on containers. When "applies=within" (the default), operations within a grouping condition must apply to the same value within a container. For example, this allows specifying conditions on two elements within the same container element. When "applies=across," conditions need not apply to the same value in the container.

Notice that the **Not** operator returns all resources that do not satisfy the specified condition, which is not the same as resources that satisfy the negation of the condition. Refer to Query 3 for an example of this distinction.

Property

The **Property** element identifies a specific, named property of a Resource. Its contents identify the named property (also known as the predicate). Its contents can be a nested property, that is, multiple property names separated by forward slashes. This syntax may navigate over multiple properties, where each property value is a resource with its own properties. This may be the same syntax used by RDF Query's "path" attribute for nested queries.

As a convenience, it may not be necessary to specify Container-related properties as part of the path, that is, Bag, Seq, Alt, and li elements are automatically navigated past.

Value

The **Value** element defines the value against which a specific property is compared. It can contain a literal string, which is compared directly against literal properties, or against a container property using one of the container operations.

In a Match Template, the **Value** element may also contain a **Variable** element, which indicates that the value is extracted from the target RDF file. The **Value** element can also specify a dt:dtype attribute that specifies the datatype of the value. The only datatype that must be explicitly specified is "dateTime," which indicates that a date comparison is to be performed on a ISO 8601 date.

Date values can also incorporate the "sysdate" keyword to indicate an operation based on the current date. Refer to Query 12 for an example.

Operation

The **Operation** element defines how the comparison is performed. RQL supports a number of predefined operations.

Literal operations operate on literal values. They include:

equals (=) performs an exact text match or numeric comparison. It will also match a resource URI.

notEquals (!=) tests for inequality.

greaterThan (>) performs the numeric comparison.

lessThan (<) performs the numeric comparison.

greaterThanOrEquals (>=) performs the numeric comparison.

lessThanOrEquals (<=) performs the numeric comparison.

like performs a substring text match.

We provide verbose forms of the various arithmetic operations for readability; this is because characters such as < require escaping within XML, which can become unwieldy.

Container operations operate on container values (Bags, Sequences, and Alternatives). They include:

contains

first

last

index(n)

sum

count

Notice that the first, last, and index() operations are only meaningful for Sequences.

Multiple Operations can be specified in a single Condition; this is useful for queries that combine container and literal operations, such as a numeric

comparison on the first entry of a Sequence. There are also two implicit shortcuts:

1. A literal operation on a container first performs an implicit "contains."
2. A container operation without a further literal operation always performs an implicit "equals."

Variable

The **Variable** element defines a substitution variable. It contains a Property element, and is used to obtain a literal value from a target RDF file. Variable elements are only found in Match Template.

Namespaces

RQL supports namespace declarations as attributes of any element. It then applies these namespaces to property values. This means that property values can use namespaces prefixes. See the examples section for several illustrations of this technique. Notice also that this is an uncommon use of namespaces; rather than applying namespace declarations to element and attribute names, it is applied to the text within the document.

Notice also that for variables, the corresponding namespace declarations must exist in the target RDF file, as opposed to the RQL file itself.

Document Type Definition (DTD) for RQL Documents

```
<!-- An RQL document contains a single Select element. -->
<!ELEMENT rdfquery (select)>

<!-- Each Select clause contains a single Condition.
 The "properties" attribute defines the information to
return as part of the result set.
 Note that the URI of each matching Resource is always
returned. -->
<!ELEMENT select (condition)>
<!ATTLIST select properties NMTOKENS #IMPLIED>
```

```
        <!-- A Condition can either directly contain an operation,
or contain a boolean grouping operator -->
                <!ELEMENT condition ( (operation+, property, value,
condition?) | and | or | not)>

        <!-- Boolean grouping operators -->
        <!ELEMENT and (condition, condition+) >

        <!-- the "applies" attribute determines whether or not the
condition within a grouping operation must
        all apply to the same value in a Collection. -->
        <!ATTLIST and applies (within | across) "within">

        <!ELEMENT or (condition, condition+) >
        <!ATTLIST or applies (within | across) "within">

        <!ELEMENT not (condition) >

        <!-- An operation defines how to compare a property to a
value -->
        <!ELEMENT operation (#PCDATA) >

        <!-- Property identifies a specific property in an RDF file.
        For container objects, any children are acceptable
        matches, and intervening Container and Description tags are
automatically navigated past. -->
        <!ELEMENT property (#PCDATA)>

        <!-- A value defines the value to which a property is
compared.  It is either a constant String, or a
        Variable whose value comes from a target RDF file.
        -->
        <!ELEMENT value (#PCDATA | variable)* >

        <!-- The value element can have a dt:type attribute
specifying its datatype -->
        <!ATTLIST value dt:type NMTOKEN #IMPLIED>
```

```
      <!-- A variable indicates a property value obtained from a
target RDF file;  it contains a Property element.  -->
      <!ELEMENT variable (property)>
```

The following are exemplary embodiments of RQL documents. The example queries may all use the following source RDF document:

```
      <?xml version="1.0"?>
      <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-
ns#"
            xmlns:hr="http://www.saba.com/hr#"
            xmlns:ewp="http://www.saba.com/ewp#"
            xmlns:ems="http://www.saba.com/ems#"
            xmlns:vCard="http://imc.org/vCard/3.0#">
         <rdf:Description
about="http:/www.saba.com/people/sally_brown">
          <vCard:N rdf:parseType="Resource">
            <vCard:Family>Brown</vCard:Family>
            <vCard:Given>Sally</vCard:Given>
            </vCard:N>
            <vCard:UID>987-65-4320</vCard:UID>
            <vCard:ROLE>Manager</vCard:ROLE>
            <vCard:ORG rdf:parseType="Resource">
              <vCard:Orgname>Development</vCard:Orgname>
            </vCard:ORG>
            <hr:Location>HQ</hr:Location>
            <hr:Reports>
              <rdf:Bag>
                <rdf:li
resource="http://www.saba.com/people/Snoopy"/>
                <rdf:li
resource="http://www.saba.com/people/Woodstock"/>
              </rdf:Bag>
            </hr:Reports>
            <ewp:competency>
              <rdf:Bag>
                <rdf:li>Java.Expert</rdf:li>
                <rdf:li>XML.Proficient</rdf:li>
```

```
            </rdf:Bag>
          </ewp:competency>
          <ewp:Interests>
            <rdf:Bag>
              <rdf:li>Java</rdf:li>
              <rdf:li>EJB</rdf:li>
              <rdf:li>COM</rdf:li>
            </rdf:Bag>
          </ewp:Interests>
          <ems:Training_Locations>
            <rdf:Seq>
              <rdf:li>San Francisco, CA</rdf:li>
              <rdf:li>San Jose, CA</rdf:li>
              <rdf:li>Los Angeles, CA</rdf:li>
              <rdf:li>Denver, CO</rdf:li>
            </rdf:Seq>
          </ems:Training_Locations>
        </rdf:Description>

        <rdf:Description
about="http:/www.saba.com/people/sally_brown" bagID="ID001">
          <ewp:competency>EJB.Advanced</ewp:competency>
        </rdf:Description>

        <rdf:Description aboutEach="#ID001">
          <ewp:attained>1999-02-25</ewp:attained>
          <ewp:provider
rdf:resource="http://www.sabanet/AllAboutJava/"/>
          <ewp:details>
            <rdf:Bag>
              <rdf:li>CBT</rdf:li>
              <rdf:li>evaluation</rdf:li>
            </rdf:Bag>
          </ewp:details>
        </rdf:Description>
      </rdf:RDF>
```

The following exemplary query ("Query 1") associated with the above source RDF document selects all managers in a department:

```
<?xml version="1.0"?>
<!DOCTYPE rdfquery SYSTEM "rql.dtd">

<rdfquery>
   <select>
      <condition
xmlns:vCard="http://imc.org/vCard/3.0#">
         <operation>equals</operation>
         <property>vCard:ROLE</property>
         <value>Manager</value>
      </condition>
   </select>
</rdfquery>
```

The following exemplary query ("Query 2")selects all developers in a department, or everyone in a development organization:

```
<?xml version="1.0"?>
<!DOCTYPE rdfquery SYSTEM "rql.dtd">

<rdfquery>
   <select>
      <condition
xmlns:vCard="http://imc.org/vCard/3.0#">
         <operation>equals</operation>
         <property>vCard:ORG/vCard:ORGNAME</property>
         <value>Development</value>
      </condition>
   </select>
</rdfquery>
```

The following exemplary query ("Query 3") selects the name and division of everyone who is not located at a headquarter location:

```
<?xml version="1.0"?>
<!DOCTYPE rdfquery SYSTEM "rql.dtd">
```

```
<rdfquery>
   <select properties="vCard:FNAME vCard:ORG"
xmlns:vCard="http://imc.org/vCard/3.0#"
xmlns:hr="http://www.saba.com/hr#">
      <condition>
         <operation>notEquals</operation>
         <property>hr:Location</property>
         <value>HQ</value>
      </condition>
   </select>
</rdfquery>
```

The following exemplary query ("Query 4") returns slightly different results, in that it also returns all resources that do not have an hr:Location property:

```
<rdfquery>
   <select properties="vCard:FNAME vCard:ORG"
xmlns:vCard="http://imc.org/vCard/3.0#"
xmlns:hr="http://www.saba.com/hr#">
      <condition>
         <not>
            <condition>
               <operation>equals</operation>
               <property>hr:Location</property>
               <value>HQ</value>
            <condition>
         </not>
      </condition>
   </select>
</rdfquery>
```

The following exemplary query ("Query 5") finds an employee named "Sally Brown":

```
<?xml version="1.0"?>
<!DOCTYPE rdfquery SYSTEM "rql.dtd">
```

```
<rdfquery>
   <select xmlns:vCard="http://imc.org/vCard/3.0#">
      <condition>
         <and applies="within">
            <condition>
               <operation>equals</operation>

<property>vCard:N/vCard:Family</property>
               <value>Brown</value>
            <condition>
            <condition>
               <operation>equals</operation>

<property>vCard:N/vCard:Given</property>
               <value>Sally</value>
            </condition>
         </and>
      <condition>
   </select>
</rdfquery>
```

The following exemplary query ("Query 6") selects everyone with a competency of "Advanced" in EJB:

```
<?xml version="1.0"?>
<!DOCTYPE rdfquery SYSTEM "rql.dtd">

<rdfquery>
   <select>
      <condition xmlns:ewp="http://www.saba.com/ewp#">
         <operation>contains</operation>
         <property>ewp:Competency</property>
         <value>EJB.Advanced</value>
      </condition>
   </select>
</rdfquery>
```

The following exemplary query ("Query 7") selects everyone who will train in San Francisco:

```
<?xml version="1.0"?>
<!DOCTYPE rdfquery SYSTEM "rql.dtd">

<rdfquery>
   <select>
      <condition xmlns:ems="http://www.saba.com/ems#">
         <operation>contains</operation>
         <property>ems:Training_Locations</property>
         <value>San Francisco, CA</value>
      </condition>
   </select>
</rdfquery>
```

The following exemplary query ("Query 8") selects everyone will train in some location in California and return to that location:

```
<?xml version="1.0"?>
<!DOCTYPE rdfquery SYSTEM "rql.dtd">

<rdfquery>
   <select properties="ems:Training_Locations"
xmlns:ems="http://www.saba.com/ems#">
      <condition>
         <operation>like</operation>
         <property>ems:Training_Locations</property>
         <value>CA</value>
      </condition>
   </select>
</rdfquery>
```

The following exemplary query ("Query 9") selects everyone whose first choice of training location is anywhere in California:

```
<?xml version="1.0"?>
<!DOCTYPE rdfquery SYSTEM "rql.dtd">
```

```
<rdfquery>
   <select properties="ems:Training_Locations"
xmlns:ems="http://www.saba.com/ems#">
      <condition>
         <operation>index(1)</operation>
         <operation>like</operation>
         <property>ems:Training_Locations</property>
         <value>CA</value>
      </condition>
   </select>
</rdfquery>
```

The following exemplary query ("Query 10") finds the manager of an employee named "Woodstock":

```
<?xml version="1.0"?>
<!DOCTYPE rdfquery SYSTEM "rql.dtd">

<rdfquery>
   <select>
      <condition xmlns:hr="http://www.saba.com/hr#">
         <operation>contains</operation>
         <property>hr:Reports</property>

<value>http://www.saba.com/people/Woodstock</value>
      </condition>
   </select>
</rdfquery>
```

The following exemplary query ("Query 11") finds all who have more than two direct reports:

```
<?xml version="1.0"?>
<!DOCTYPE rdfquery SYSTEM "rql.dtd">

<rdfquery>
   <select>
      <condition xmlns:hr="http://www.saba.com/hr#">
         <operation>count</operation>
```

```
                        <operation>greaterThan</operation>
                        <property>hr:Reports</property>
                        <value>2</value>
                    </condition>
                </select>
            </rdfquery>
```

The following exemplary query ("Query 12") finds all who have an advanced competency rating in EJB, with the competency ratings obtained from evaluations.

```
<?xml version="1.0"?>
<!DOCTYPE rdfquery SYSTEM "rql.dtd">

<rdfquery>
   <select xmlns:ewp="http://www.saba.com/ewp#">
      <condition>
         <operation>equals</operation>
         <property>ewp:competency</property>
         <value>EJB.Advanced</value>
         <condition>
            <operation>contains</operation>
            <property>ewp:details</property>
            <value>evaluation</value>
         </condition>
      </condition>
   </select>
</rdfquery>
```

The following exemplary query ("Query 13") finds everyone hired in the past month:

```
<?xml version="1.0"?>
<!DOCTYPE rdfquery SYSTEM "http://dlipkin/rql.dtd">

<rdfquery>
   <select xmlns:hr="http://www.saba.com/hr#"
xmlns:dt="urn:w3-org:xmldatatypes">
```

```
            <condition>
               <operation>greaterThan</operation>
               <property>hr:StartDate</property>
               <value dt:type="dateTime">sysdate-31</value>
            </condition>
         </select>
      </rdfquery>
```

Information Distributor Implementation

The following is an exemplary implementation embodiment of Info Distributor in the platform of the invention. The implementation has two components:

1. DatabaseMR – a Java class that implements a Metadata Repository (MR) on top of a relational database. This class provides utility methods to be invoked by Import Agents, Match Agents, and Delivery Agents.

2. RQL parser – a Java class that implements the RQL query language. It parses an RQL document and executes the query using the DatabaseMR.

In an embodiment, DatabaseMR implements the MR interface, that is, it provides the ability to import an RDF document, return the value of an RDF property, and perform a metadata match.

DatabaseMR uses a database schema containing the following tables:

MR_sources – contains URI references to each imported document

Column	Datatype	Description
id	number	Primary key
source_URI	varchar2(1024)	URI of imported document

MR_triples_base - stores the actual data of all RDF triples from imported RDF documents.

Column	Datatype	Description
uri_ref	number	Foreign key to MR_sources table
rdf_property	varchar2(1024)	Property values
rdf_resource	varchar2(1024)	Resource values
rdf_object	varchar2(1024)	Object values

In addition, there is a view called **MR_triples** defined as

CREATE VIEW MR_triples AS (SELECT rdf_property, rdf_resource, rdf_object FROM MR_triples_base)

This view allows other data sources to also be manipulated by the MR, as described below.

As an example, the following RDF document:

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:dc="http://purl.org/dc/elements/1.1/"
xmlns:schedule="http://www.saba.com/RDF/schedule/1.0#">
  <rdf:Description resource="http://dlipkin/class1">
     <dc:title>HTML Fundamentals</dc:title>
     <schedule:startDate>1998-12-07</schedule:startDate>
  </rdf:Description>
</rdf:RDF>
```

appears as the following data:

rdf_resource	rdf_property	rdf_object
http://dlipkin/class1	http://purl.org/dc/elements/1.1/title	HTML

		Fundamentals
http://dlipkin/class1	http://www.saba.com/RDF/schedule/1.0#startDate	1998-12-07

The methods of DatabaseMR are implemented as follows:

importRDF()

The importRDF() method imports RDF data. It uses W3C's open-source RDF parser, SiRPAC (http://www.w3.org/RDF/Implementations/SiRPAC/) to generate triples from an RDF document.

This algorithm followed by this method is:

1. See if this document has already been imported. If so, delete all triples resulting from the previous import.

2. Insert the key for this document into MR_sources.

3. Invoke SiRPAC to parse the document and generate triples, using Java code similar to the following:

```
        private void generateTriples(Reader r, String key) throws
ImportException
        {
           r = (Reader) new RDFReader(r);

           InputSource source = new InputSource(r);
           source.setSystemId(key);

            RDFConsumer consumer = (RDFConsumer) new
DatabaseMRConsumer(this);

           mSirpac.setRDFSource(source);
           mSirpac.setStreamMode(mUSE_STREAMING_PARSER);
           mSirpac.register(consumer);
           mSirpac.fetchRDF();
        }
```

where DatabaseMRConsumer is a callback class invoked by SiRPAC that simply invokes the insertTriple() method of DatabaseMR:

```
private class DatabaseMRConsumer implements RDFConsumer {

   private DatabaseMR mMR;

   public DatabaseMRConsumer(DatabaseMR theMR)
   {
      mMR = theMR;
   }

   public void start (DataSource ds) {}
   public void end   (DataSource ds) {}

   public void assert (DataSource ds, Resource predicate,
Resource subject, RDFnode object) {
      mMR.insertTriple(predicate.toString(),
subject.toString(), object.toString());
    }
 };
```

4. Insert each triple into the MR_triples_base table using a prepared statement of the form:

```
INSERT INTO MR_triples_base(id, uri_ref, rdf_property,
rdf_resource, rdf_object) VALUES(MR_sequence.nextval, ?, ?, ?, ?)
```

5. Commit the transaction.

match()

The match() method takes a MatchDescriptor specifying a Match Agent and Delivery Agent and performs a match. It uses the following algorithm:

1. Extract the RDF query and target RDF document from the MatchDescriptor.

2. Parse the query using RQLParser.

3. Execute the query by invoking the getResources() method on the root Operator returned by RQLParser. Pass in the target RDF document as an argument, and obtain a result Vector of matching resource Strings.

4. Construct a MatchResultSet of the query results.

5. Dispatch the query results to the Delivery Agent.

getProperty()

The getProperty() method returns the value for a specific property stored in the MR. It does this by invoking a SQL statement of the form:

```
SELECT rdf_object FROM MR_triples WHERE rdf_resource = ? AND
rdf_property = ?
```

Database schema

The database schema used has two main advantages:

1. Simplicity. All RDF data is stored in a single table and all SQL is written to read and write to this table.

2. Support for non-RDF data. It is simple to cast non-RDF data into this format so that existing or legacy data can be queried by the DatabaseMR using RQL.

So, for example, for the following example data stored in an "invoices" table:

id	last_updated	customer
1	10-JAN-99	Ford
2	25-FEB-99	Cisco

The view used by the MR can be augmented as followed to incorporate this data:

```
create view invoice_date_triples as
select   'last_updated' "rdf_property",
      ('invoice#' || id) "rdf_resource",
      to_char(last_updated, 'YYYY-MM-DD') "rdf_object"
      from test_invoices;

create view invoice_customer_triples as
select   'customer' "rdf_property",
      ('invoice#' || id) "rdf_resource",
      customer "rdf_object"
      from test_invoices;

drop view MR_triples;
create view MR_triples as
  (select rdf_property, rdf_resource, rdf_object from
invoice_date_triples)
   union
  (select rdf_property, rdf_resource, rdf_object from
invoice_customer_triples)
   union
  (select rdf_property, rdf_resource, rdf_object from
MR_triples_base);
```

This will result in the following additional triples being available from the MR:

rdf_resource	rdf_property	rdf_object
invoice#1	last_updated	10-JAN-99
invoice#1	customer	Ford
invoice#2	last_updated	25-FEB-99
invoice#2	customer	Cisco

The disadvantage to this schema is that it is not normalized and stores a tremendous amount of duplicate data. Many values for rdf_resource and rdf_property will be duplicated, since the same resource will have a number of properties, and property names will come from a well-known set.

RQLParser

RQLParser parses an RQL document and builds an execution plan for the query. The plan consists of a tree of Java classes called "Operators," where each Operator is responsible for returning a Vector of matching resources.

The Operator interface is defined as follows:

```
public interface Operator
{
/**
 * An operator knows how to return a Vector of matching
resource values
 * (typically URIs).
 * @param conn JDBC connection to the MR
 * @param targetRDF Target RDF file.
 * @return Vector of matching resources
 * @exception SQLException Thrown on a database error
 */
   public Vector getResources(Connection conn, String
targetRDF) throws SQLException, ParseException;

} /* Operator */
```

A variety of Operators are provided, each of which is responsible for handling different RDF constructs or RQL operations. Some of the available Operators are:

AndOperator – implements the "and" boolean operator. It contains an array of child Operators. It calls getResources() on each one, then constructs a result Vector of the resource that are present in each and every child.

OrOperator – implements the "or" boolean operator. It contains an array of child Operators. It calls getResources() on each one, then constructs a result Vector of the resource that are present in any child.

SimpleOperator – an abstract class that contains a property string, a value string, and a child Operator. It is the superclass for both SingleOperator and ContainerOperator.

SingleOperator – a SimpleOperator that handles basic expressions, ie equals or notEquals. It executes a SQL query of the form:

```
SELECT DISTINCT rdf_resource FROM (SELECT * FROM MR_triples
WHERE rdf_property = ?) WHERE rdf_object [operation] ?
```

The value for [operation] is provided by the concrete subclass. Available subclasses include:

EqualsOperator
NotEqualsOperator
GreaterThanOperator
LessThanOperator
LikeOperator

The value used to match the rdf_object can either be provided as hard-coded text in the RQL document, or it can be defined as a variable containing a propertyName. In this case, a metadata match is performed, using the target RDF document as the source for the property value.

ContainerOperator – a SimpleOperator that operates on an RDF container (a Bag, Seq, or Alt). It contains a child operator that it executes to return a set of

generated resources representing the RDF container. It then executes a SQL query of the form:

```
SELECT rdf_resource FROM MR_triples WHERE rdf_property = ?
AND rdf_object = ?
```

where each rdf_object is set to one of the child resources.

Additionally, there is an OperatorRegistry class where each Operator is registered with the RQL operation it supports.

RQLParser uses the following algorithm and methods for generating the execution plan:

1. parse():

Parse the RQL document using a standard XML parser to obtain the resulting DOM tree.

Navigate to the main condition node and call parseCondition() on it.

2. parseCondition():

If the condition is a boolean, call parseBoolean().

Otherwise, call parseOperation().

3. parseBoolean():

Obtaining each child node and recusively calling parseCondition() on each one.

Create the appropriate Operator for the boolean (AndOperator, OrOperator, NotOperator) with the children obtained by calling parseCondition().

4. parseOperation():

Obtain the operation, property, and value nodes.

Extract the text values of these nodes, and call createOperator() with these values.

5. createOperator():

a. Use the OperatorRegistry to obtain the Java class of the Operator responsible for this operation.

b. Use Java reflection to create a new instance of this Operator class, passing in the appropriate parameters.

Agents

Agents are implemented as clients of the DatabaseMR class.

For example, a simple ImportAgent will pass its text RDF argument to the importRDF() method:

```
public class SimpleImportAgent implements ImportAgent
{

  private MR mMR = null;
  public SimpleImportAgent(MR theMR)
  {
    mMR = theMR;
  }

  public void importRDF(String rdf) throws ImportException
  {
  Reader r = (Reader) new StringReader(rdf);

  /* this import has a unique key so it can never be
overridden by
  subsequent imports */
  String key = "generated" + System.currentTimeMillis();
  mMR.importRDF(r, key);
  } /* importRDF */

} /* SimpleImportAgent */
```

A simple MatchAgent will take an RQL document and a DeliveryAgent as parameters, and invoke the match() method:

```
public class SimpleMatchAgent implements MatchAgent
{
   private MR mMR = null;
   private DeliveryAgent mDA = null;
   private MatchDescriptor mMD = null;

   public SimpleMatchAgent(MR theMR, String rql,
DeliveryAgent theDA)
   {
     mMR = theMR;
     mDA = theDA;
     mMD = new MatchDescriptor(rql, "", (MatchHandler)
theDA);
   }

   public void match() throws MatchException
   {
     mMR.match(mMD);
   } /* match */

} /* SimpleMatchAgent */
```

A simple DeliveryAgent prints the RDF document containing the matching resources to System.out:

```
public class SimpleDeliveryAgent implements MatchHandler
{
   public void deliver(MatchResultSet mrs) throws
DeliveryException
   {
     String xml = mrs.toXML();
     System.out.print(xml);
   }
```

```
} /* SimpleDeliveryAgent */
```

BEST MODE

As indicated earlier in **Figure 3**, the architecture of a preferred embodiment of the present invention adopts a three-tier model. Referring now to **Figure 17**, the various types of computer hardware and computer software used in a preferred embodiment at the present time are depicted in greater detail. In **Figure 17**, a tier 1 user workstation 1701 and a tier 1 dedicated user personal computer (PC) 1703 are connected electronically to a tier 2 web server 1707 via the Internet 1709. **Figure 17** also shows a tier 1 user smart phone 1705 directly connected to a tier 2 application server 1711, such as the SABA Business Platform. And the tier 2 applications server 1711 may be connected to a tier 3 database management system 1713, additional external SABA systems 1715, external third party systems 1717 and/or third party knowledge management systems 1719.

The user workstation 1701 can be a Sun® Ultra5™ workstation and the user PC 1703 can be any general purpose PC. Note that the list of tier 1 devices presented in this preferred embodiment are not conclusive. Other tier 1 user devices could be WebTV or other Personal Assistant Devices (PDAs). A Sun E250™ dual processor server can be used as a development/test system running the Sun® Solaris® operating environment, Oracle® 8I. A single processor Sun E250™ server can be used for the SABA Business Platform, as a Sun E4500™ dual processor, an IBM NetFinity 7000™ quad processor with a Microsoft® NT™ server and a Hitachi Shared Disk array. The workstation 1701 and the PC 1703 can interface to the tier 2 SABA Business Platform through the Internet 1709 using a standard Internet browser such as Internet Explorer™. The tier 3 database can be located on an Oracle 8I® server, a SQL server, or Informix. The Sun E250™ dual processor server can interface with the external third party systems 1717 via third party system specific adapter plugs. The Sun E250™ dual processor server also interfaces with external SABA systems 1715 via SABA

exchange. Finally, the Sun E250™ dual processor server can also interface with the tier 3 database management system 1713 located on the Oracle 8I® server.

Referring again to **Figure 17**, the tier 2 applications server 1711 is expanded to illustrate the SABA Business Platform (Platform) of the present invention. In **Figure 17**, the Platform contains an Interface Server 1721, an Information Server 1723, an Interconnect Server 1725, and a Business Server 1727. In a preferred embodiment, all of these Servers 1721, 1723, 1725, and 1727 may physically reside on the same hardware platform (such as a UNIX box or a Microsoft™ NT™ platform), or each server may reside on a separate hardware box, or any combination of servers and hardware boxes. Each of the servers has included a JAVA Virtual Machine™ and the related runtime support.

In a preferred embodiment, the business server 1727 embodies the containers which incorporate all of the business logic, common business objects, SABA core objects, and a database driven framework for generating notifications and for triggering periodic events based on context sensitive attachments. The business server 1727 communicates with each of the other servers within the Platform using the XML protocol (1727, 1729, and 1731). The Business Server 1727 also communicates with the database management system 1713. In communicating with the interface server 1721, the business server 1727 first generates a XML message 1729 and transmits it to the interface server 1721. The interface server 1721 then performs style sheet transformations on the XML using XSL or XSLT to translate the XML message into the particular Applications Programming Interface (API) language required to communicate with a particular user. For example, if a particular user is accessing the Platform via a workstation 1701 or a PC 1703, the Interface Server 1721 can convert the XML 1729 into HTML 1735 and communicate with the user through a web server 1707 via the Internet 1709. The Interface Server 1721 can also convert the XML into other protocols such as WAP/WML 1737 to communicate with Personal Data Assistants (PDAs) such as cell phones 1705, Palm Pilots™, or other such wireless devices. Since the interface that is generated between the Platform and the various user interfaces is dictated by the set of style sheets generated in the Interface Server 1721, the same

core business logic can be leveraged to communicate across a number of different user interfaces.

The Interconnect server 1725 uses XML to communicate with both the Information server 1723 and the Business server 1727 and is responsible for all connectivity external to the Platform. Externally, the Interface Server 1721 may communicate with third party systems such as SAP™ accounting or personnel packages, Oracle™ financial or human resources, other SABA Platforms 1715, and generally any external system to which a portion of the Interconnect facilities may be connected. The Interconnect server 1725 comprises SABA interconnect 1739 which is essentially a backplane into which cards or interconnect services can be plugged. Examples of these cards or interconnect services can be an event monitor 1741, exchange registry, node manager 1747, connectors, accessor 1743, or subscribers 1745. Each of these cards or interconnect services leverage the services provided by the SABA interconnect backplane 1739 for communicating between the cards themselves and for providing more sophisticated services to third party systems 1717.

A preferred embodiment of the Platform may interconnect with a third party system 1717 having, for example, an Oracle human resources (HR) database 1749 and an Oracle financial database 1751. The third party system 1717 has a third party interconnect backplane 1753 with similar cards or interconnect services. The third party interconnect backplane 1753 connects to the third party databases 1749 and 1751 via system specific adapters 1755. These system specific adapters 1755 differentiate between different types of databases such Oracle, SAP, or PeopleSoft and feed into the standardized Platform framework so information can be exchanged. An example of information that can be exchanged includes HR information. When a new employee is added to or terminated from the third party HR system database 1749, the monitor 1757 located on the third party interconnect backplane 1753 notifies the subscriber 1745 located on the SABA interconnect backplane 1739 via XML 1759. The accessor 1743 on the SABA interconnect backplane 1739 can then access the new employee data via XML. The Interconnect server 1725 then performs style sheet transformations to

convert the XML into the Platform's native format and transmits that data to the Business server 1727 which then updates the database management system 1713. This data connection can be set to be automatic or with modifications.

In a preferred embodiment, the Interconnect server 1725 also embodies a workflow and notification scheme. For example, if a group of students signed up for a class through the Platform and later the class time changes, the Platform can detect this change and initiate a workflow to obtain all the names of the students from the database management system 1713 and send an email to them notifying them of the change. Thus, the interconnect server 1725 can provide real-time, in-order, reliable updating of data, financial transactions, or management of human capital between the Platform and third party systems 1717.

The Interconnect server 1725 can also be used to synchronize the Platform with other external SABA systems 1715. For example, the Platform can publish a catalog and based on permissions that are set, the catalog can be subscribed to by some other external SABA systems 1715. Whenever changes are made to the catalog, the external SABA systems 1715 can monitor that change and obtain an update immediately. The Interconnect server 1725 can also connect to SABA private learning networks which are connected to SABA public learning networks via SABA Exchange. For example, a third party such as Ford Automotive may have a SABA system allowing them to exchange catalog or class course information via the interconnect server 1725.

The Information Server 1723, communicates with the Interconnect server 1725 and the Business Server 1727 via XML. The Information Server 1723 also communicates directly with the database management system 1713 for query and storage of metadata 1733. The Information server 1723 focuses on queries and distributed queries and keeping track of information about other pieces within the Platform. The Information Server 1723 can also leverage the Interconnect server 1725 to connect to a third party knowledge management system 1719 that generates information via the SABA Interconnect backplane 1739. For example, a third party may have a third party Interconnect backplane 1761 connected to a Knowledge Management System 1719 which monitors and exchanges data with

the Platform via XML. The Information Server 1723 contains Import, Match and Delivery agents to resolve and generate information requests; Match templates to match metadata; and template-based services that respond to information requests and are capable of rendering their results in XML; and Finders - metadata driven, template-based query builders that generate optimized SQL queries in the native SQL language of the particular database involved.

Having described the invention in terms of a preferred embodiment, it will be recognized by those skilled in the art that various types of general purpose computer hardware may be substituted for the configuration described above to achieve an equivalent result. Similarly, it will be appreciated that arithmetic logic circuits are configured to perform each required means in the claims for performing the various features of the rules engine and flow management. It will be apparent to those skilled in the art that modifications and variations of the preferred embodiment are possible, such as different computer systems may be used, different communications media such as wireless communications, as well as different types of software may be used to perform equivalent functions, all of which fall within the true spirit and scope of the invention as measured by the following claims.